Usages of Composition Search Tree in Web Service Composition

Lakshmi.H.N[a], Hrushikesha Mohanty [b]

[a]PhD Scholar, University of Hyderabad, Hyderabad, Contact: hnlakshmi@gmail.com

[b]Professor, University of Hyderabad, Hyderabad.

The increasing availability of web services within an organization and on the Web demands for efficient search and composition mechanisms to find services satisfying user requirements. Often consumers may be unaware of exact service names that's fixed by service providers. Rather consumers being well aware of their requirements would like to search a service based on their commitments(inputs) and expectations(outputs). Based on this concept we have explored the feasibility of I/O based web service search and composition in our previous work[6]. The classical definition of service composition ,i.e one-to-one and onto mapping between input and output sets of composing services,is extended to give rise to three types of service match: Exact,Super and Partial match. Based on matches of all three types, different kinds of compositions are defined: Exact,Super and Collaborative Composition. Process of composition,being a match between inputs and outputs of services,is hastened by making use of information on service dependency that is made available in repository as an one time preprocessed information obtained from services populating the registry. Adopting three schemes for matching for a desired service outputs, the possibility of having different kinds of compositions is demonstrated in form of a Composition Search Tree. As an extension to our previous work, in this paper, we propose the utility of Composition Search Tree for finding compositions of interest like leanest and the shortest depth compositions.

1. INTRODUCTION

Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web.As growing number of services are being available, searching the most relevant web service fulfilling the requirements of a user query is indeed challenging.

Various approaches can be used for service search, such as,searching in UDDI, Web and Service portals.The techniques for searching web services can be divided into two categories: discovery and composition. By service composition, we mean making of a new service(that does not exist on its own) from existing services.It can be useful when we are looking for a web service for given inputs and desired outputs and there is no single web service satisfying the request[6].

Most of the existing algorithms[8,2,3,4,5,7,1] for service composition construct chains of services based on exact matches of input/output parameters to satisfy a given query.However,the making of a chain fails at a point when inputs of a succeeding(I^S) service does not match exactly with the outputs (O^P) of a preceeding service.

To alleviate this problem, in [6] we propose a Collaborative Composition among such partially matching services for satisfying a desired service outputs, by making match criteria flexible. In addition to exact match we allow partial as well as super match for conditions $O^P \subset I^S$ and $O^P \supset I^S$ respectively.Partial match is of our interest and in [6] we have shown the possibility of successful service composition by collaboration of services that make only partial matches. The process of service composition is visualized as a Composition Search Tree [6] that arranges services in levels showing the way service compositions can be made to meet the user requirements.Our approach[6] results to a scalable implementation for use of RDBMS, a well proven technology.

Here, as an extension to our previous work, we propose the utility of Composition Search Tree

for finding optimal service compositions.

We define two such optimal compositions -

- *LeanestComposition* - A service composition that requires minimum number of web services to satisfy a given query.
- *ShortestDepthComposition* - A service composition satisfying a given query that has minimum depth in the Composition Search Tree.

The remainder of this paper is organized as follows. In Section 2 we essay the related work. In Section 3 we give a brief account of our previous work - service composition process using three modes of composition.Also, we explain Composition Search Tree with an example. Section 4 describes the utility of Composition Search Tree. Algorithms for finding Leanest Composition and Shortest Composition are explained in this section. We conclude our work in Section 5.

2. RELATED WORK

In this section, we survey current efforts related to web services composition, built on relational databases, considering input/output parameters of web services.A web service, ws, has typically two sets of parameters from $\{P_i\}$ as set of inputs ws^I and set of outputs ws^O.Conventionally two services ws_i and ws_j are said to be composable iff $ws_i^O = ws_j^I$,i.e,ws_j receives all the required inputs from outputs ws_i has[6].

Recently, many researchers have utilized techniques in relational database to solve the service composition problem. Lee et al. [5,7] proposed a scalable and efficient web services composition system based on a relation database system.They pre-compute all possible web service compositions,by applying multiple joins on the tables maintained and store them to be used later for web service composition search. PSR system supports web services having single Input and Output parameters.

Zheng et al.[1] put forward a new storage strategy for web services which can be flexibly extended in relational database. A matching algorithm SMA is proposed that considers the semantic similarity of concepts in parameters based on WordNet.Based on their storage strategy they propose an algorithm:Fast-EP, for searching service composition.

The current techniques based on relational database are constrained by usage of multiple joins as well as malady of exact match of input and output parameters.In [6] we propose an approach to overcome these difficulties. The criteria for matching is relaxed for partial matching allowing several services to collaborate and provide a desired service. In the current work we further extend the utility of Composition Search Tree for finding optimal service compositions.

3. I/O MATCH BASED SERVICE COMPOSITION

In this section, we summarize our previous work[6] in which we propose an approach to extend the classical definition of service composition.We first define the problem statement, followed by the various service composition modes proposed, then give a brief explanation of the composition process and finally explain the Composition Search Tree with an example.

3.1. Problem Statement

Given a service registry $R = \langle P, W \rangle$ and a query $Q = \langle Q^I, Q^O \rangle$,we need to find set of web services, $WS \subseteq W$, $WS = \{ws_1, ws_2, , ws_n\}$, such that services in WS can be composed to obtain Q^O, $\{ws_1^O \cup ws_2^O \cup .... \cup ws_n^O\} \supseteq Q^O$,where

- P is a set of parameters,$P = \{P_1, P_2, , P_n\}$.
- W is a set of web services in the registry, $W = \{ws_1, ws_2, ..., ws_n\}$.
- ws_i^O is a set of output parameters of web service ws_i.
- $\{ws_1^O \cup ws_2^O \cup .... \cup ws_n^O\}$ is the union of output parameters of $ws_1^O, ws_2^O, ..., ws_n^O$.
- $Q^I \subset P$ is set of initial input parameters

- $Q^O \subset P$ is set of desired output parameters

3.2. Service Composition Types

Given a registry $R = \langle P, W \rangle$, any desired set of output parameters,$D^O \subset P$, can be satisfied by possibly many compositions. To generate such compositions we start by matching the output of services in the registry,ws_i^O,with D^O and classify the services on their composability as Exact,Super and Partial. We can readily define two types of compositions - Exact Composition and Super composition, from the Exact Composable and Super Composable services as follows -

1. **Exact Composition (EC)**: Exact Composition is a composition obtained by using a web service that is Exactly Composable with D^O,i.e. ,$ws_i^O = D^O$, where $ws_i \in W$.Such a composition would require additional input parameters (RI_{EC}^I) than those specified in Q^I given by,

$$RI_{EC}^I = ws_i^I - Q^I$$

where ws_i^I is input parameters of web service ws.There can be many services in W that are Exactly composable with D^O and one of them is chosen in each level to be solved further.

2. **Super Composition (SC)**: Super Composition is a composition obtained by using a web service that is Super Composable with D^O, i.e. , $ws_i^O \supset D^O$, where $ws_i \in W$.The additional input parameters required by such a composition (RI_{SC}^I) is given by,

$$RI_{SC}^I = ws_i^I - Q^I$$

where ws_i^I is input parameters of web service ws. One of the many services in W that are Super composable with D^O is chosen at each level to be solved further.

Most of the existing algorithms for service composition construct chains of services based on Exact Matches of input/output parameters to satisfy a given query.However,this approach fails when the available services satisfy only a part of the input/output parameters in the given query.This shortcoming motivated us to define a new type of composition - Collaborative Composition, that is obtained by using a set of partial composable services. We define Collaborative Composition as -

Collaborative Composition (CC): Collaborative Composition is a composition obtained by using a set of partial composable services , WS, that can collaboratively satisfy the desired set of output parameters D^O, i.e ,there exists a set of services WS_{CC} , such that

$$WS_{CC} \subset W \ , \ WS_{CC} = \{ws_1, ws_2, ..., ws_n\}$$

$$\text{where } ws_i^O \subset D^O, \forall ws_i \in WS_{CC}$$

$$\text{and } \{ws_1^O \cup ws_2^O \cup ... \cup ws_n^O\} \supseteq D^O$$

There can be many such service sets that satisfy D^O.The additional input parameters required (RI_{CC}^I) to execute the services in WS_{CC} is given by

$$RI_{CC}^I = WS_{CC}^I - Q^I - WS_F^O$$

where WS_{CC}^I is collective input parameters required by the set WS_{CC}, i.e ,

$$WS_{CC}^I = \left\{ws_1^I \cup ws_2^I \cup .... \cup ws_n^I\right\}$$

and WS_F is a set of services such that

$$WS_F \subseteq WS_{CC} \ , \ WS_F = \{ws_1, ws_2, ..\}$$

$$\text{such that } \forall ws_j \in WS_F, ws_j^I \subseteq Q^I.$$

3.3. Composition Process

In this section we describe the process of generating service compositions satisfying a given query, proposed in our previous work[6].The steps involved in composition process is as below -

- **Search for Matching Services** : The composition process starts with searching

for services in the registry whose output parameters match with the required output parameters as specified in the user query(Q^O).

- **Classify the Compositions** : The many compositions satisfying Q^O are classified as Exact Composition, Super Composition and Collaborative Composition.We then choose one of the many possible compositions in each type,in each level,and create three child nodes,(Left,Middle and Right), representing each type: Exact, Super and Collaborative Composition,respectively.

- **Solve for Additional Input Parameters**: In the next level these compositions are solved for the additional input parameters required,to those provided as input parameters in the query(Q^I).The matching compositions are categorized on their composability mode and one of the compositions of each type is chosen to be solved further in the next level for additional input parameters required.

- **Repeat process untill all compositions are found**: The process is repeated recursively until the tree explores all compositions satifying the given user query.

3.4. Composition Search Tree

In order to visualize the composition process and to find all possible compositions that satisfy a given user query we construct a Composition Search Tree[6]. The Composition Search Tree supports querying for optimal service compositions such as Leanest Composition and Shortest Depth Composition.

The structure of a node in *CompositionSearchTree* is given by Backus Naur Form(BNF) in Fig 1.The abbreviations used in BNF are described in Table 1.

Figure 1. BNF of a CST Node

<CST Node>::=<CST Data> <CST Pointer> <Node Type>

<CST Data>::=<WS> <NWS> <D^O> <Composition Type>
<WS>::={WS ID}*
<NWS>::= <Integer>
<D^O>::= {Parameter Symbol}*
<CompositionType>::= <Exact> | <Super> | <Collaborative> |<NIL>

<CST Pointers> ::= <Parent Node> [<Left Child>] [<Middle Child>] [<Right Child>]
<Parent Node> ::=<Pointer to CST Node> (* Present except for Root Node *)
<Left Child> ::=<Pointer to CST Node whose Composition Type = Exact>
<Middle Child> ::=<Pointer to CST Node whose Composition Type = Super>
<Right Child> ::=<Pointer to CST Node whose Composition Type = Collaborative>

<Node Type> ::= <Root> | <Internal> | <UnSolvable> | <Solution>

Table 1
Abbreviations used in BNF

Abbreviation	**Description**
WS	Set of web services participating in Composition
NWS	Number of web services used
D^O	Desired set of output parameters

The *CompositionSearchTree* has 4 types of nodes as described below -

1. **Root Node** : A CST node from where the composition process begins,having the following special properties -
   - $\langle WS \rangle = \emptyset$
   - $\langle NWS \rangle = 0$
   - $\langle D^O \rangle = Q^O$

- $\langle CompositionType \rangle = NIL$
- $\langle ParentNode \rangle = NULL$

2. **Internal Node** : A CST node that represents a composition (Exact,Super or Collaborative) satisfying D^O of its Parent Node.Every internal node of the *CompositionSearchTree* has utmost 3 *ChildNodes*, a $LeftChildNode$ representing *ExactComposition*, a $MiddleChildNode$ representing *SuperComposition* and a *RightChildNode* representing *CollaborativeComposition*. Although there may be many compositions in each type : Exact, Super and Collaborative, that satisfy D^O of the Parent Node, we propose to choose one of the compositions in each type for every internal node and hence limit the number of children to three.Note that the Root Node is also an Internal Node with special properties as explained before.

3. **UnSolvable Node** : A leaf node that cannot be solved further since D^O of such a node does not have matching compositions in *ServiceComposabilityTable*. These type of nodes have the following special properties -

   - $\langle CompositionType \rangle = \langle Exact \rangle | \langle Super \rangle | \langle Collaborative \rangle$
   - $\langle D^O \rangle = \{ParameterSymbol\}^*$
   - $\langle LeftChild \rangle = NULL$
   - $\langle MiddleChild \rangle = NULL$
   - $\langle RightChild \rangle = NULL$

4. **Solution Node** : A leaf node that need not be solved further since D^O of such a node is $\emptyset$.These type of nodes represent compositions solving the given user query and have the following special properties -

   - $\langle CompositionType \rangle = \langle Exact \rangle | \langle Super \rangle | \langle Collaborative \rangle$
   - $\langle D^O \rangle = \emptyset$
   - $\langle LeftChild \rangle = NULL$
   - $\langle MiddleChild \rangle = NULL$
   - $\langle RightChild \rangle = NULL$

Every node of the *CompositionSearchTree* stores the composition that satisfies desired output parameters of its parent node (WS),number of web services used for composition(NWS) and set of additional input parameters required by the composition (D^O).

Each node in the Composition Search Tree has utmost 3 *ChildNodes*, a $LeftChildNode$ representing *ExactComposition*, a $MiddleChild\ Node$ representing *SuperComposition* and a *RightChildNode* representing *CollaborativeComposition*.

Fig 2 depicts Composition Search Tree using the Web services in Table 2 to construct the tree for a query with $Q^I = \{Date, City\}$ and $Q^O = \{HotelName, FlightInfo, CarType, TourCost\}$.

The process for Composition Search Tree construction is given below :

1. Create a *RootNode* that has desired output parameters equal to the output parameters specified in the query, i.e $D^O = Q^O$,initialize the number of web services used in composition NWS to 0 and set of web services participating in composition as empty set, $WS = \emptyset$.

2. *Insert* the *RootNode* to *LiveNodesQ*.

3. *Delete* a *LiveNode* from *LiveNodesQ* and set it as the *CurrentNode*.

4. Find services that match with D^O of the *CurrentNode*.

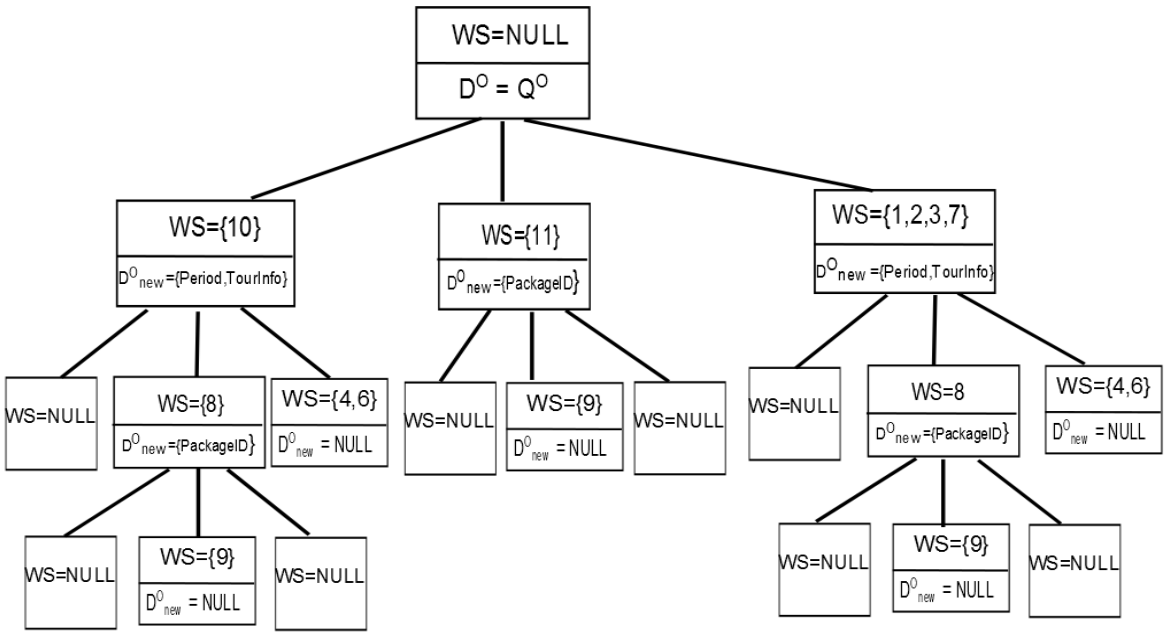
5. Classify these services according to their match type.

6. Find different compositions that satisfy D^O from these services based on their match type as -

Table 2
Example Web Services

WS No	Service Name	Input Parameters	Output Parameters
ws1	HotelBooking	Period,City	HotelName,HotelCost
ws2	AirlineReservation	Date,City	FlightInfo,FlightCost
ws3	TaxiInfo	Date,City	CarType,TaxiCost
ws4	DisplayTourInfo	HotelName, FlightInfo,CarType	TourInfo
ws5	TaxiReservation	CarType,Date,City	TaxiCost
ws6	TourPeriod	Date,City	Period
ws7	TourCost	TourInfo	TourCost
ws8	AgentPackage	PackageID	Period,TourInfo
ws9	TourPackages	Date,City	PackageID
ws10	TourReservation	Period,TourInfo	HotelName,FlightInfo, CarType,TourCost
ws11	PackageDetails	PackageID	HotelName,Hotelcost, FlightInfo,FlightCost, CarType,TaxiCost, TourCost

Figure 2. Example CST



(a) If a service ws having an Exact Match with D^O is available in registry,create a $LeftChildNode$ for the $CurrentNode$,store ws and update NWS as $NWS = NWS+1$.Calculate the additional input parameters required,to execute ws, as $R^I_{EC} = ws^I - Q^I$,where ws^I is input parameters of web service ws. R^I_{EC} is the new set of desired output parameters that need to be satisfied, i.e.,$D^O = R^I_{EC}$.If $D^O \neq \emptyset$ then insert the $LeftChildNode$ to $LiveNodesQ$,otherwise mark the $LeftChildNode$ as $SolutionNode$ and $Insert$ a copy of the node to $Solutions$. Make the $LeftChildNode$ point to its $ParentNode$.

(b) If a service ws having an Super Match with D^O is available in registry,create a $MiddleChildNode$ for $CurrentNode$,store ws and update NWS as $NWS = NWS+1$.Calculate the additional input parameters required,to execute ws, as $R^I_{RC} = ws^I - Q^I$,where ws^I is input parameters of web service ws.R^I_{SC} is the new set of desired output parameters that need to be satisfied,i.e.,$D^O = R^I_{RC}$. If $D^O \neq \emptyset$ then insert the $MiddleChildNode$ to $LiveNodesQ$, otherwise mark the $MiddleChildNode$ as $SolutionNode$ and $Insert$ a copy of the node to $Solutions$. Make the $MiddleChildNode$ point to its $Parent\ Node$.

(c) Among services that have Partial match find a set of services that can collaboratively satisfy D^O.If such a set, WS, is available, create a $RightChildNode$ for the $CurrentNode$,store WS and update NWS as $NWS = NWS + |WS|$.Calculate the additional input parameters required,if any,to execute the services in WS, as $R^I_{CC} = WS^I - Q^I - WS^O_F$ where WS^I is collective input parameters required by the set WS, i.e , $WS^I = \{ws^I_1 \cup ws^I_2 \cup .... \cup ws^I_n\}$ and WS_F is a set of services such that $WS_F \subseteq WS, WS_F = \{ws_1, ws_2, ..\}$ $|\forall ws_j \in WS_F, ws^I_j \subseteq Q^I.R^I_{CC}$ is the new set of desired output parameters that need to be satisfied, i.e.,$D^O = R^I_{CC}$.If $D^O \neq \emptyset$ then insert the $RightChildNode$ to $LiveNodesQ$,otherwise mark the $RightChildNode$ as $SolutionNode$ and $Insert$ a copy of the node to $Solutions$. Make the $Right\ ChildNode$ point to its $ParentNode$.

(d) If D^O cannot be satisfied by any of the above 3 cases then mark $CurrentNode$ as $UnsolvableNode$.

7. *Delete* a $LiveNode$ from $LiveNodesQ$ and set it as the $CurrentNode$.
8. Find services that match with WS of the $CurrentNode$.
9. Repeat steps 5 to 8 untill the $LiveNodesQ$ becomes empty.

4. UTILITY OF COMPOSITION SEARCH TREE

As discussed earlier the Composition Search Tree not only finds all possible compositions satifying a given query but can also be utilized for querying for optimal service compositions.In this paper,we define two such optimal service compositions.

4.1. Leanest Composition

A service composition that requires minimum number of web services to satisfy a given query is called Leanest Composition.The Leanest Composition is an optimal composition in that it uses least number of services possible for service composition.The procedure for searching a Leanest

Composition in Composition Search Tree is given in Algorithm 1.

4.2. Shortest Depth Composition

A service composition satisfying a given query that has minimum depth in the Composition Search Tree is called Shortest Depth Composition. The Shortest Depth Composition is an optimal composition in that it has least depth in Composition Search Tree.The procedure for searching a Shortest Depth Composition in Composition Search Tree is given in Algorithm 2.

4.3. Observations

The following observations can be made from the Algorithms for Leanest Composition and Shortest Depth Composition -

- **Observation 1:** A *SolutionNode* representing *ShortestDepthComposition* also represents a *LeanestComposition* if it appears at a *Level i* that is equal to NWS.

  Rationale: This observation can be reduced from Line number 10 in Algorithm 2 and Line number 13 in Algorithm 1. Line number 10 in Algorithm 2 always returns the first *SolutionNode* as obtained in the breadth first search of the Composition Search Tree.If this *SolutionNode* has the property that it appears at a *Level i* that is equal to NWS, then this node will be returned as *SolutionNode* from Line number 13 in Algorithm 1, since this node will have the least NWS among all *SolutionNodes*.

- **Observation 2:** A *SolutionNode* represents a *LeanestComposition* iff there are no other *SolutionNodes* in the Composition Search Tree that has a lesser NWS than this *SolutionNode*.

  Rationale: This observation can be reduced from Line numbers 13 and 20 in Algorithm 1. These statements search for the *SolutionNode* with the least NWS and hence the algorithm always returns a *SolutionNode* that has the least NWS.

5. CONCLUSION

This paper is an extension to our previous work in [6]. In [6] the scope of composition is widened defining possibly three modes of service composability: Exact,Partial and Super. Based on composability of all three types and sequencing them differently, *CompositionSearchTree* explores all possible compositions for a given requirement. In the current work, we propose the utility of *CompositionSearchTree* for finding optimal service compositions like *Leanest Composition* and *ShortestDepthComposition*.

The set of web services returned by the algorithms in sections 4 and in [6] implicitly includes the final composition plan when Exact and Super composition or a combination of the two are involved,given by a chain of service calls from the *SolutionNode* till the *RootNode*.However,a composition plan needs to be derived from the set of services whenever the composition includes Collaborative Composition.In the future work we would like to work on an algorithm that generates a composition plan specifying the order of execution for services participitating in a Collaborative composition.Since our system explores all possible compositions for a given requirement,we would like to include a monitoring component that monitors execution of composition and suggests an alternative composition in case of any service failure.

REFERENCES

1. C. Zheng, W. Ou, Y. Zheng, and D. Han. Efficient Web Service Composition and Intelligent Search Based on Relational Database. International Conference on Information Science and Applications (ICISA), 2010, 1-8.
2. H.N. Talantikite et al., Semantic annotations for web services discovery and composition, Computer Standards Interfaces,31(6),1108-1117. Elsevier B.V, 2009.
3. I.B. Arpinar et al., Ontology-driven web

Algorithm 1: Searching Leanest Composition

Input: *CompositionSearchTree*
Output: *SolutionNode* for Leanest Composition

```
1  CurrentNode = RootNode of CST , Level = 0
2  CArr and NLCArr are arrays of ChildNodes
   // LCN is Leanest Composition SolutionNode
3  LCN = NULL
4  minNWS = ∞ // Current minimum number of services
5  Insert all ChildNodes of CurrentNode to CArr
6  while CArr is not empty do
7      Level = Level + 1
8      foreach ChildNode in CArr do
9          CN=ChildNode
10         if CN is an SolutionNode then
               // NWS is number of web services used
11             if Level = NWS then
12                 if NWS < minNWS then
                       // CN is the shortest composition
13                     LCN=CN
14                     return [LCN]
15                 else
16                     if LCN ≠ NULL then
                           // LCN already present
17                         return [LCN]
18             else
19                 if NWS < minNWS then
                       // A shorter composition
20                     LCN=CN
21         if CN is an UnsolvableNode then
22             Go To 7
23         else // CN is an UnResolvedNode
24             Insert all ChildNodes of CN to NLCArr
25     CArr=NLCArr
26     CArr=0
27 if LCN ≠ NULL then
28     return [LCN]
29 else
30     return [NULL] // Composition Not Found
```

Algorithm 2: Searching Shortest Depth Composition

Input: *CompositionSearchTree*
Output: *SolutionNode* for Shortest Depth Composition
`// Initialization Steps`
CurrentNode = *RootNode* of *CST*
Level = 0
CArr and *NLCArr* are arrays of *ChildNodes*
Insert all *ChildNodes* of *CurrentNode* to *CArr*
while *CArr is not empty* **do**
    Level = *Level* + 1
    foreach *ChildNode in CArr* **do**
        CN=*ChildNode*
        if *CN is an SolutionNode* **then**
            return *[CN]*
        if *CN is an UnsolvableNode* **then**
            Go To 6
        else*CN* is an *UnResolvedNode*
            Insert all *ChildNodes* of *CN* to *NLCArr*
    CArr=*NLCArr*
    CArr=0
return *[NULL]* `// Composition Not Found`

services composition platform. Inf. Syst. E-Business Management, 2005, 3(2):175199.
4. J. Gekas, M. Fasli, Automatic Web Service Composition Based on Graph Network Analysis Metrics, In Proceedings of the International Conference on Ontology, Databases and Applications of Semantics (ODBASE). Agia Napa, Cyprus, 2005, pp. 1571-1587.
5. J. Kwon, K. Park, D. Lee, S. Lee, PSR: Precomputing Solutions in RDBMS for Fast Web services Composition Search, in: Proceedings of the 2nd International Conference on Web Services, Salt Lake City, Utah, USA, 2007 pp. 808815.
6. Lakshmi, H.N.and Mohanty H., RDBMS for service repository and composition,2012 Fourth International Conference on Advanced Computing (ICoAC),13-15 Dec. 2012.
7. Lee D., Kwon J., Lee S., Park S., Hong B.Scalable and efficient web services composition based on a relational database (2011) Journal of Systems and Software, 84 (12) , pp. 2139-2155.
8. S. Hashemian, F. Mavaddat, A graph-based framework for composition of stateless web services, In Proceedings of ECOWS06, IEEE Computer Society, Washington, DC, 2006, pp. 7586.

Lakshmi H N is currently a PhD scholar, SCIS(School Of Computer and Information Sciences), University of Hyderabad, Hyderabad. She is working as Associate Professor, CVR College of Engineering, Hyderabad.

She received her M.S. in Software Systems from BITS Pilani and B.Tech degree from BMS College of Engineering, Bangalore University. Her areas of interest include Web Services, Data Structures, etc.

Hrushikesha Mohanty is a Professor in SCIS(School Of Computer and Information Sciences), University of Hyderabad, India. Worked previously in Electronics Corporation of India Limited, Hyderabad.

Took M.Sc(Mathematics) at Utkal University and completed his Ph.D(Computer Science) from IIT Kharagpur. He has over 85 research publications in refereed International Journals and Conference Proceedings..